

MORSL: Minimal-Overhead Rank-Based Predictor with Summation-Free Correction and Lazy Access

Toru Koizumi¹, Masanari Mizuno¹, Soma Nishida¹,
Kanata Abe², Tomoaki Tsumura¹, Ryota Shioya²

¹Nagoya Institute of Technology, ²The University of Tokyo

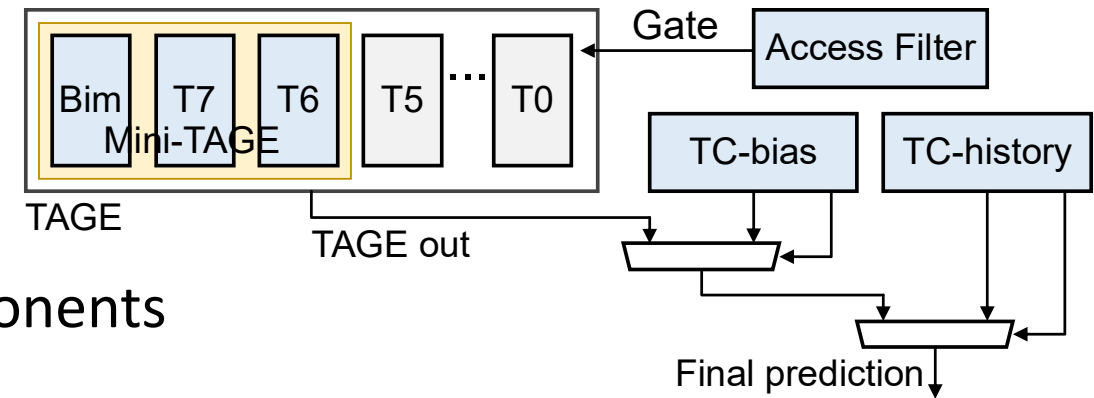


- Key requirements for realistic branch predictors:
 - Accuracy, high throughput, low latency, low energy overhead
- TAGE-SC: among the most accurate branch predictors.
 - TAGE achieves high prediction accuracy, but at high energy cost.
 - Statistical Corrector (SC) has implementation challenges: multiple table reads, long latency, and high energy consumption.
- Proposed branch predictor: MORSL
 - TAGE-based high-accuracy branch predictor
 - Dynamic access gating to reduce energy while preserving accuracy
 - Summation-free correction for TAGE to improve accuracy

MORSL structure

3/20

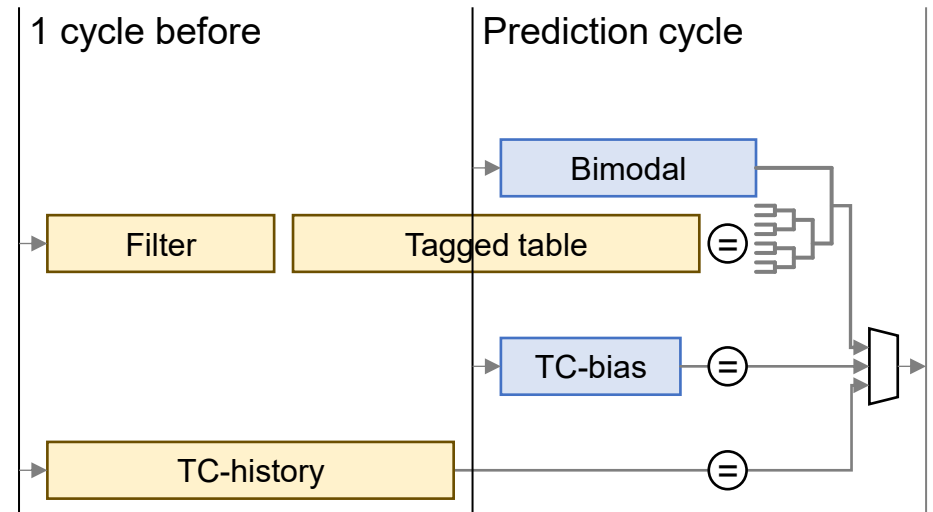
- 9-component rank-based TAGE
 - Bim + 8 tagged components
- Allocation-Guided Access Filter
 - gates accesses to 6 out of 8 tagged components before they are read → energy reduction
- Two override-style tagged correctors
 - TC-bias: captures statistical biases
 - TC-history: captures auxiliary-history correlations, mostly from BrIMLI



MORSL timing

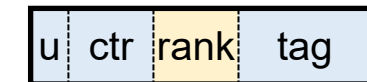
4/20

- P1 latency = P2 latency = 286 ps (< 1 cycle)
 - If we develop a predictor with a 2-cycle P2 latency, VFS decreases by > 0.005 even if P1 predicts perfectly due to the increase in misprediction penalty.
 - 4.5 MPKI with a 10-cycle penalty $\approx$ 5.0 MPKI with a 9-cycle penalty
 - Speculative fetch using P1 would address this, but CBP-NG rules do not allow it.
- TAGE cannot predict in 1 cycle
 - Ahead indexing for history components
 - Non-ahead access to PC-indexed tables
 - TAGE base predictor and TC-bias
 - Non-ahead tag matching $\ominus$
 - TAGE tagged component, TC-history and the filter



TAGE in MORSL (1/3)

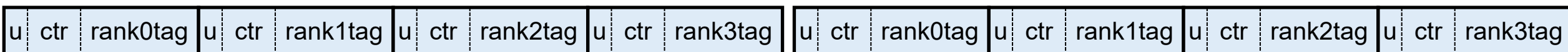
- High throughput: Predicting an unaligned 32-instruction block
- Low effective latency: Ahead indexing
- Energy-oriented design: narrow SRAM word width
 - Rank-based prediction
 - Generating up to 4-rank predictions
 - Direct-mapped
 - Rank information is encoded in the entry.
 - No extra candidate read for missing history



MORSL entry, 16 bits

way0

way1

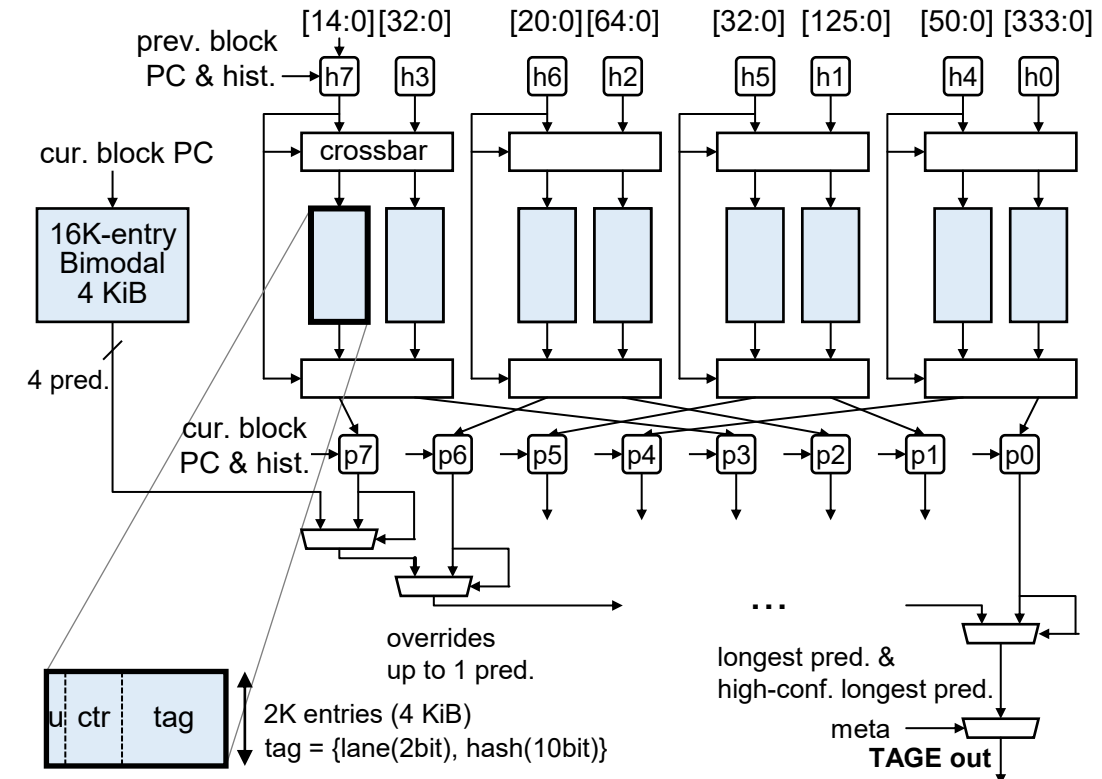


The cookbook TAGE^[1] entry, >100 bits

TAGE in MORSL (2/3)

6/20

- Base predictor: bimodal
 - predicts for 4 ranks $\rightarrow$ reads and outputs 4 bits
- Eight tagged predictors
 - 2:2 bank interleaving
 - override up to 1 bit per component
 - use 5-, 7-, 11-, 17-, 33-, 65-, 126-, and 334-block histories
 - second-order arithmetic + geometric + make longest history length very long
 - 1 bit/block for long history (h0-h3) and 3 bits/block for short histories (h4-h7)
 - the meta predictor selects between the longest-match prediction and HC-pred^[1]

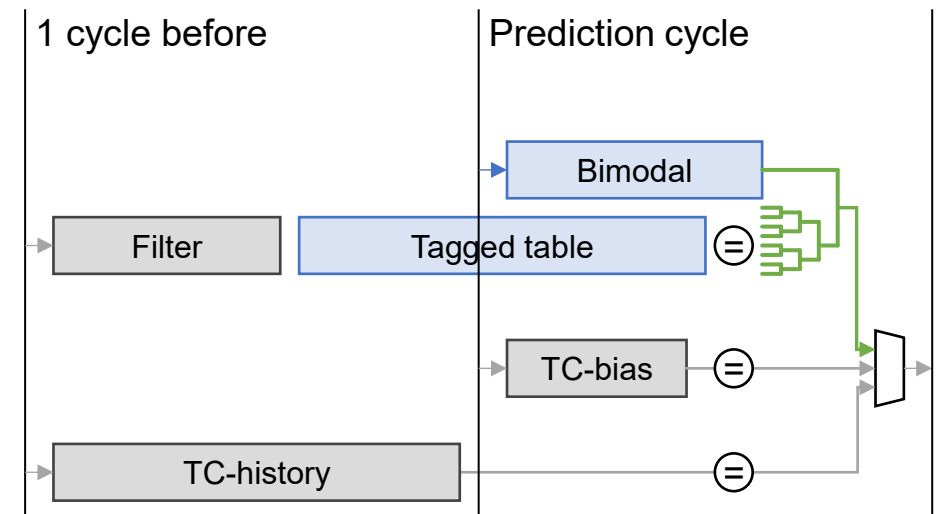


[1] A. Sez nec, TAGE: an engineering cookbook. HAL Inria, RR-9561 (2024).
<https://hal.science/hal-04804900>

TAGE in MORSL (3/3) — timing

7/20

- Bim: non-ahead indexing
 - Ahead indexing of PC-indexed predictors is challenging.
- Tagged components: ahead indexed, with non-ahead tag checking
 - Since we can check tags using latest PC, the accuracy penalty is ~1%.



Access Filter — background

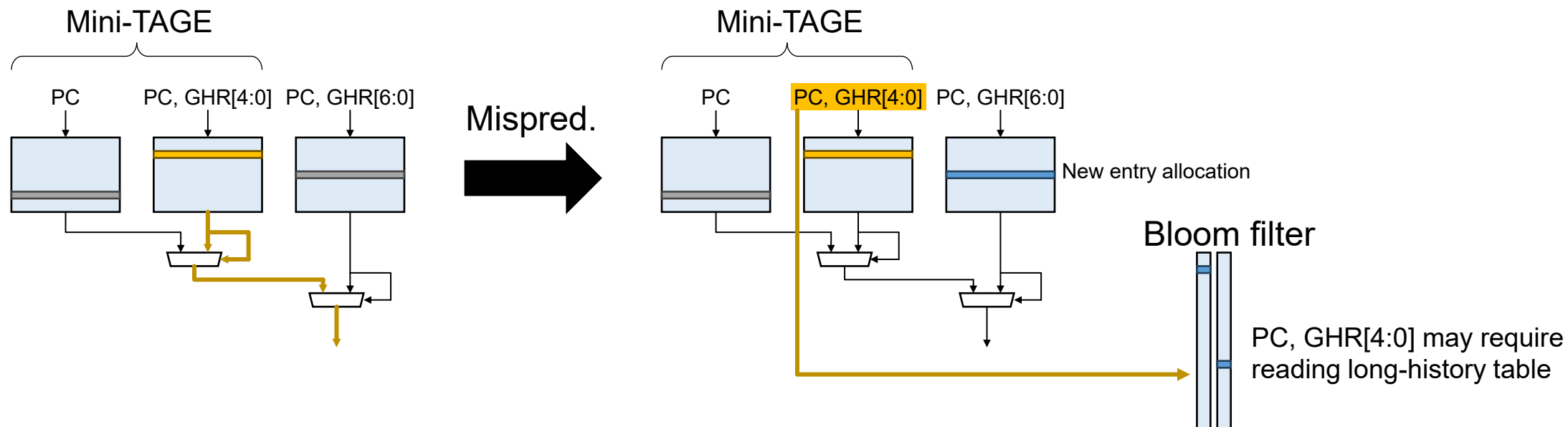
8/20

- Motivation: Gating long-history components can reduce energy.
 - Many branches can be predicted by using only PC or short history.
 - In such a situation, accessing long-history components wastes energy.
- Challenge: How can we infer this?
 - We need a small-capacity mechanism to infer that no long-history entrie exists.
 - Using long history results in poor storage efficiency.
 - We can use only short histories.
- Although these conflict with each other, TAGE allocation rules can be exploited.

Allocation-Guided Access Filter

9/20

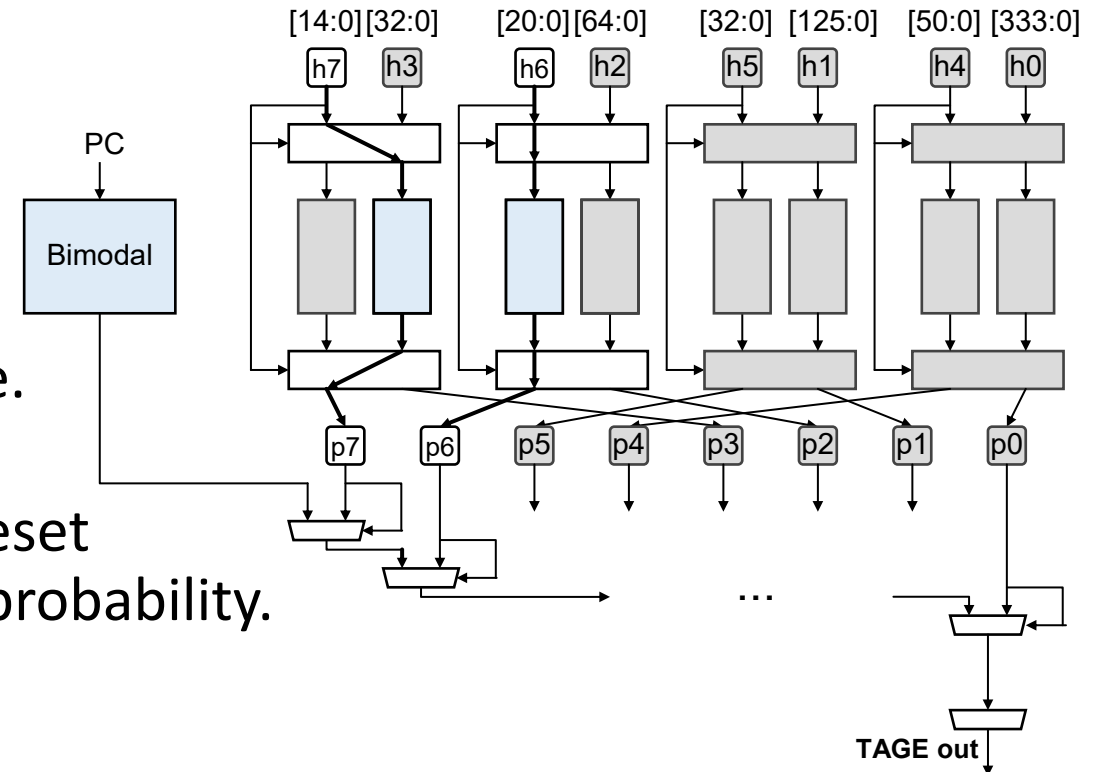
- Our approach: Exploit TAGE's allocation rule.
 - TAGE allocates no entries to a long-history component until a short-history table mispredicts. → TAGE allocation is conservative evidence that short histories were insufficient.
 - We record the allocation using a Bloom filter.



AG Access Filter — implementation

10/20

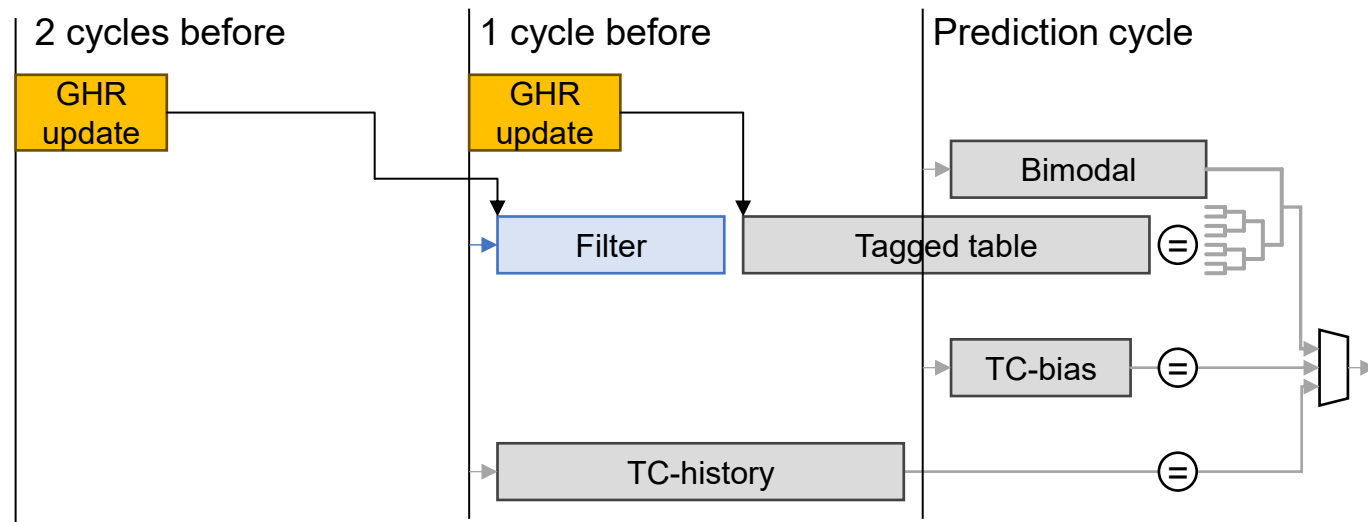
- The AG access filter gates accesses to 6 out of 8 logical components.
 - Due to the 2:2 bank interleaving, the remaining 2 logical components must read their predictions from 2 dynamically-selected physical banks.
- Random reset feature
 - The filter gradually becomes conservative.
 - thereby losing filtering opportunities
 - When mini-TAGE correctly predicts, we reset the corresponding entries with very low probability.
 - This restores the filtering opportunities.



AG Access Filter — timing

11/20

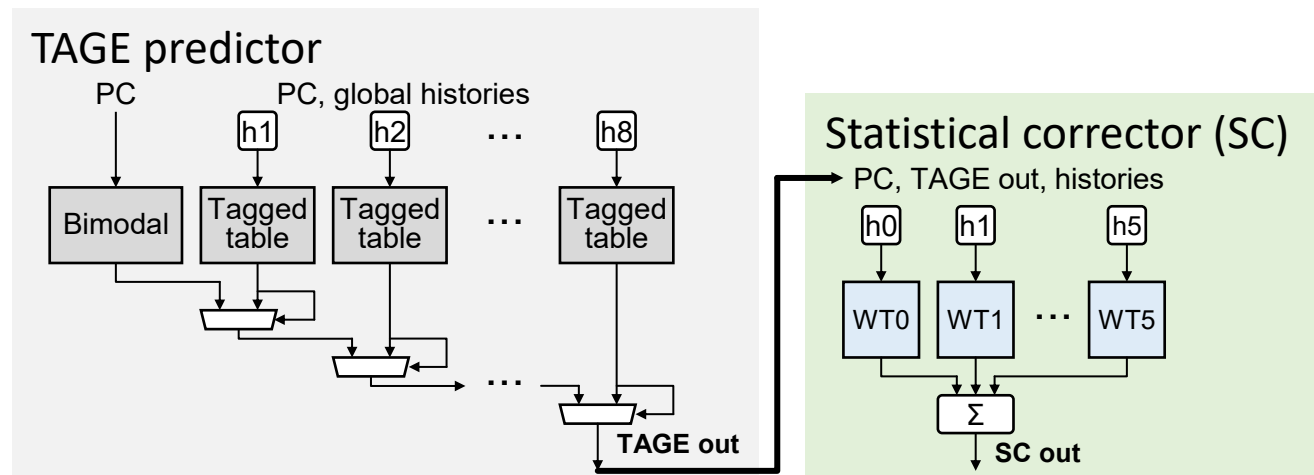
- AG Access Filter must decide before tagged-component accesses.
 - The tagged-component accesses are 1-block ahead.
- Our approach: use 1-block ahead PC and 2-block ahead history
 - We decide whether to apply the filter in parallel with the history update.
 - We cannot use 1-block ahead history due to the timing constraints.
 - We observed virtually no impact on accuracy or filtering rate.
 - We can safely ignore the missing history.



Statistical Corrector (SC) and its problem

12/20

- SC corrects patterns that TAGE struggles with.
 - TAGE-SC is among the most accurate branch predictors.
- Problems:
 - SC needs to read multiple tables. ☹️
 - SC tables that use TAGE outputs lengthen prediction latency, ☹️
 - and speculative operation to mitigate this consumes substantial energy. ☹️
- Challenge:
 - How can we correct TAGE at a low cost?



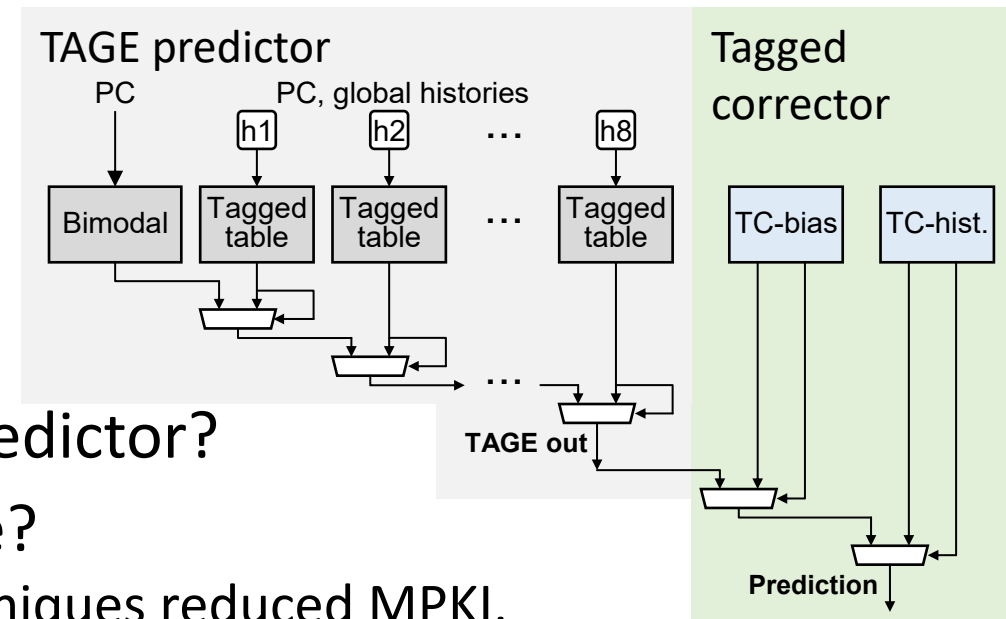
Tagged Corrector

13/20

- Our solution: two tagged correctors (TC)
 1. TC-bias: Correcting statistical biases
 2. TC-history: Capturing auxiliary history correlations
- TC is an override-style predictor.
 - It is similar to a loop predictor.
 - No reads from multiple tables are required. 😊
 - Only a small MUX is added to the critical path. 😊
 - No speculative operation is needed. 😊
- Future work 1: adding cost-effective meta predictor?
- Future work 2: multiple histories in one table?
 - Preliminary experiments showed that these techniques reduced MPKI.

The entry structure

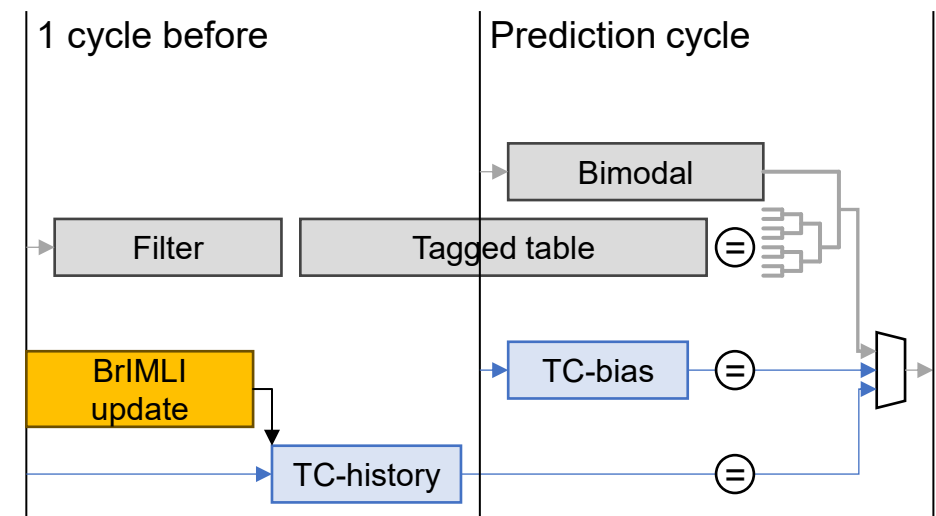
tag	ctr	conf
-----	-----	------



Tagged Corrector — timing

14/20

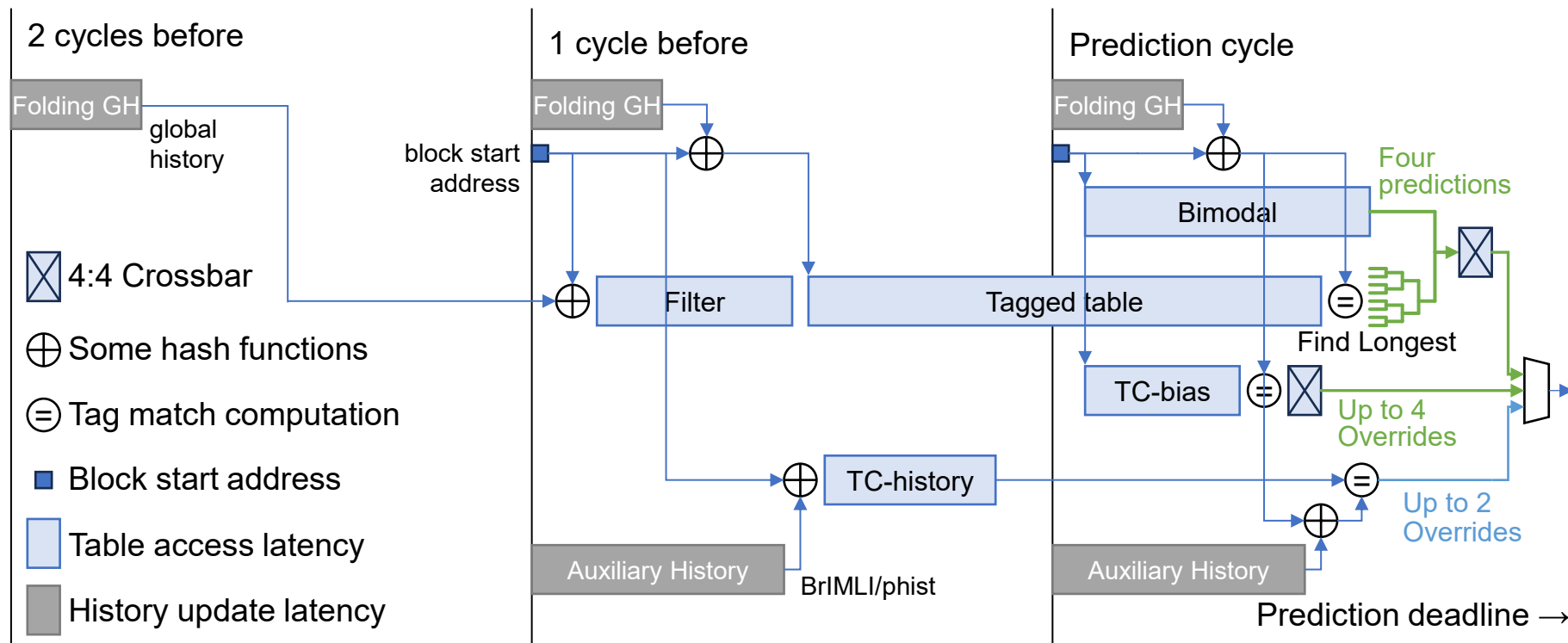
- TC-bias: non-ahead indexed
 - PC-indexed components should not be ahead indexed
- TC-history: 1-block ahead indexed, non-ahead tag checked
 - The accuracy loss from ahead pipelining is negligible (~ 0.002 MPKI)
- Updating BrIMLI has high latency
 - A backwardness decision ($\text{targetPC} < \text{branchPC}$) requires a comparator



The 37.5 KiB MORSL structure with timing

15/20

- Non-ahead: Bimodal (4 KiB), TC-bias (0.5 KiB)
- 1-block ahead: Tagged tables (32 KiB), TC-history (0.5 KiB)
- 1-block ahead PC + 2-block ahead history: AG Access Filter (0.5 KiB)

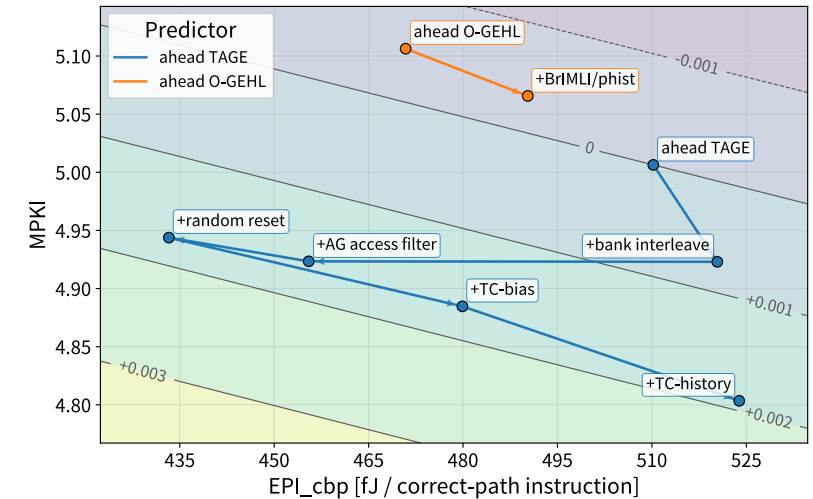


- We evaluated the 37.5 KiB MORSL using HARCOM on 168 training traces.
 - 1M-instructions warm-up; 40M-instructions measurement (or until the trace end)
 - HARCOM assumes a 5 nm process.
- We use a 36 KiB ahead TAGE as the VFS baseline. (to avoid revealing the true VFS)
 - 5.006 MPKI, 510 fJ/ins
- Perceptrons?
 - 37.25 KiB ahead O-GEHL: 5.106 MPKI, 471 fJ/ins (-0.0003 VFS vs. ahead TAGE)
 - Slightly worse accuracy, but energy efficient due to the small number of SRAMs.
 - There is virtually no difference in VFS. Note that perceptron-based predictors suffer from the overhead associated with Bloom filters under CBP-NG rules.
 - Although we proposed improvements to TAGE-based predictors, perceptron-based predictors have their own distinct strengths.

Evaluation results

17/20

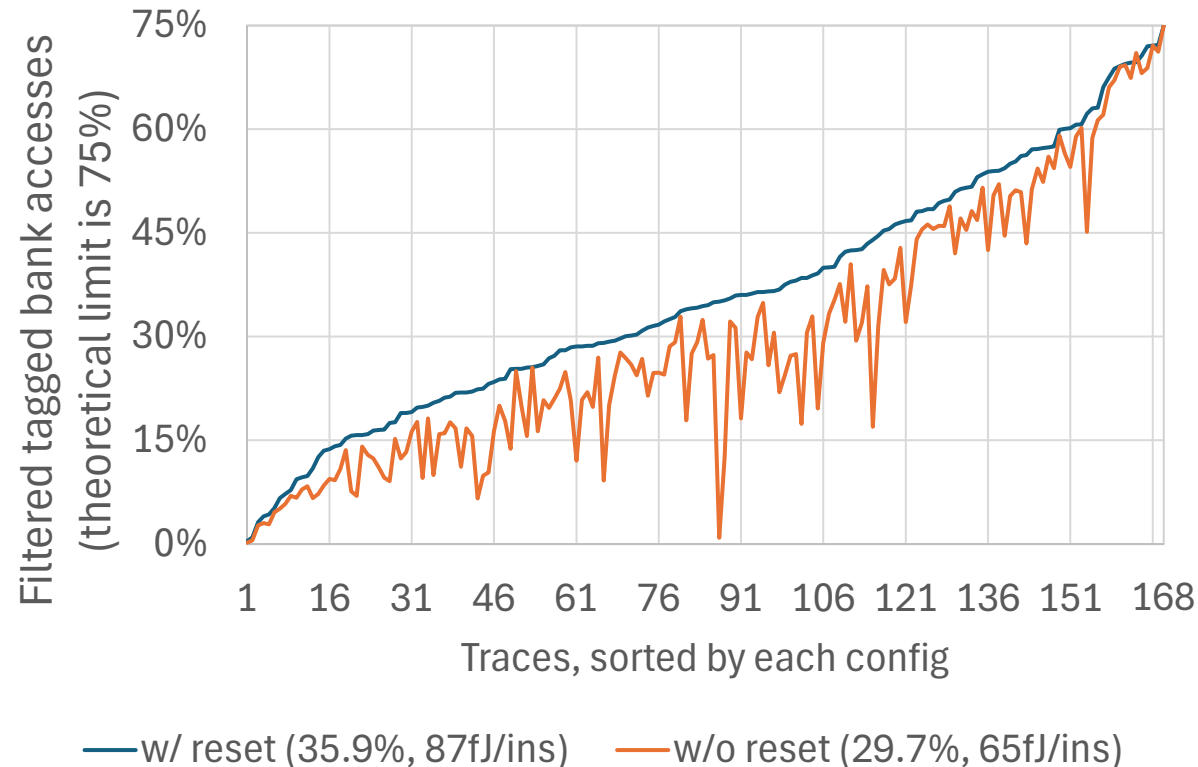
- We use “pt” to denote 10000x VFS.
 - e.g., gshareN_ahead 0.9674VFS → 9674pt
 - Rule of thumb: 1 pt $\approx$ 0.01 MPKI $\approx$ 7 fJ/ins
- MORSL: 20.8pt higher than naive ahead TAGE
 - 2:2 bank interleaving: +10 fJ/ins for 0.083 MPKI reduction [5.9 pt] (from prior work)
 - AG access filter: 65 fJ/ins reduction [9.0 pt]
 - Random reset: additional 22 fJ/ins reduction, but +0.021 MPKI [1.5 pt]
 - TC-bias: 0.059 MPKI reduction with 47 fJ/ins [1.5 pt]
 - TC-history: 0.081 MPKI reduction with 44 fJ/ins [2.9 pt]



Evaluation — Alloc.-Guided Access Filter

18/20

- AG Access Filter suppresses accesses in a wide range of applications.
- Resets significantly restore filtering opportunities in several applications.

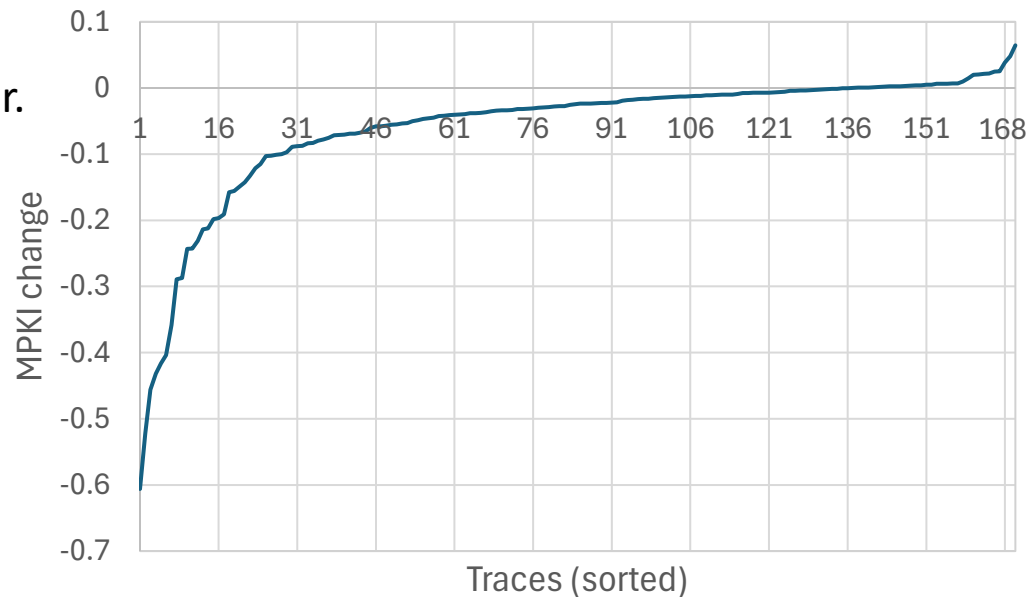


Evaluation — Tagged Corrector

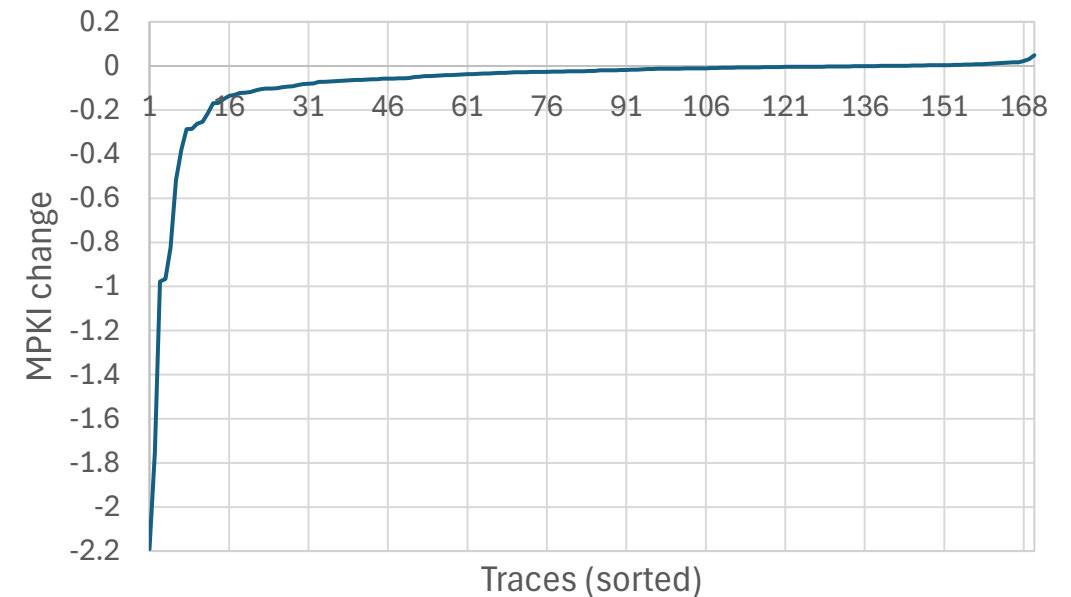
19/20

- TC-bias and TC-history
 - slightly improve TAGE across a wide range of application
 - substantially improve it in some applications

TC-bias (-0.059 MPKI)



TC-history (-0.081 MPKI)



- We propose MORSL, a cost-effective high-accuracy branch predictor.
 - High throughput: 8.647 ins/cycle (IPC_{cbp} , prediction throughput)
 - Energy efficiency: 524 fJ/ins (EPI_{cbp} , energy per predicted instruction)
 - 15.1 mW at 3.33 GHz
 - High accuracy: 4.803 MPKI
 - Low effective latency: 286 ps
- Key features of MORSL:
 - Allocation-guided access filter: reduces energy via gating long-history components
 - Tagged corrector: low-latency correction via summation-free tagged overriding, capturing statistical biases and IMLI correlations
 - I will discuss when IMLI corrects TAGE in ISCA Main Session 3D. Please come check it out!

Appendix: Parameters of O-GEHL

21/20

- Provides 4-rank prediction for an unaligned 64-instruction block
- Bloom filter (4KiB): one hash function
- Bias (5KiB): 2K 4x 5-bit weight, 2x multiplier, non-ahead indexed
- History (28KiB): 7 2K 4x 4-bit weight, ahead indexed + shuffling by XL
 - XL: missing path information (previous block end at rank #0/#1/#2/#3 or fall-through)
 - no extra candidate read; just shuffling among 4 weight sums
 - chooses 1/4/7/10/15/20/33-block or 4/10/20/33/60/94/117-block history
- 4:4 shuffling by XB
 - XB: a part of the PC, equalizing the imbalance of rank usage
- Alias monitor (0.25KiB): 2K 1-bit partial tag
 - indexed by the 33-block history

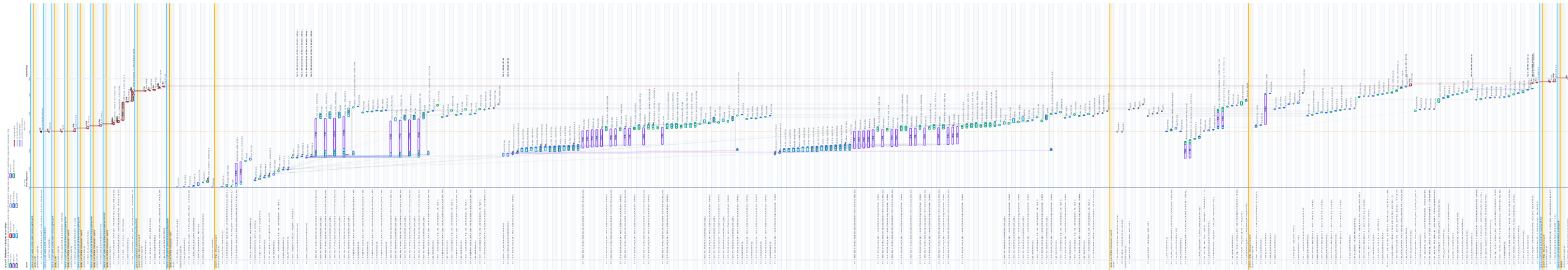
Appendix: Parameters of TAGE in MORSL

22/20

- Provides 4-rank prediction in an unaligned 32-instruction block
- Bim (4KiB): 4K 4x 2bc, non-ahead indexed
 - indexed by $PC[13:2] \wedge PC[25:14]$; folding is important since target PCs often align
- History (32KiB): 8 2K {10b tag, 2b XB-shuffled lane, 1b useful, 3bc}
 - ahead indexed by $PC.reverse \wedge fold11 \wedge (fold10 \ll 1) \wedge fold10$ -like functions
 - Using two folded histories is important (0.03MPKI). The fold10 is reused for calculating tag.
 - no extra candidate read; just non-ahead tag checked
 - one-tag-per-SRAM-word design exploiting TAGE's pseudo-skewed associativity
 - 2:2 bank interleaving (h0-h4, h1-h5, h2-h6, h3-h7)
 - [h0-h7, h1-h6, h2-h5, h3-h4] seems interesting, but this is the worst pairing!
 - short history: 15/21/33/51-bit history information for 5/7/11/17-block
 - long history: 33/65/126/334-bit history information for 33/65/126/334-block
 - Using 3-/1-bit per block seems too overengineered; 2-bit per block would be more practical.

Major timings

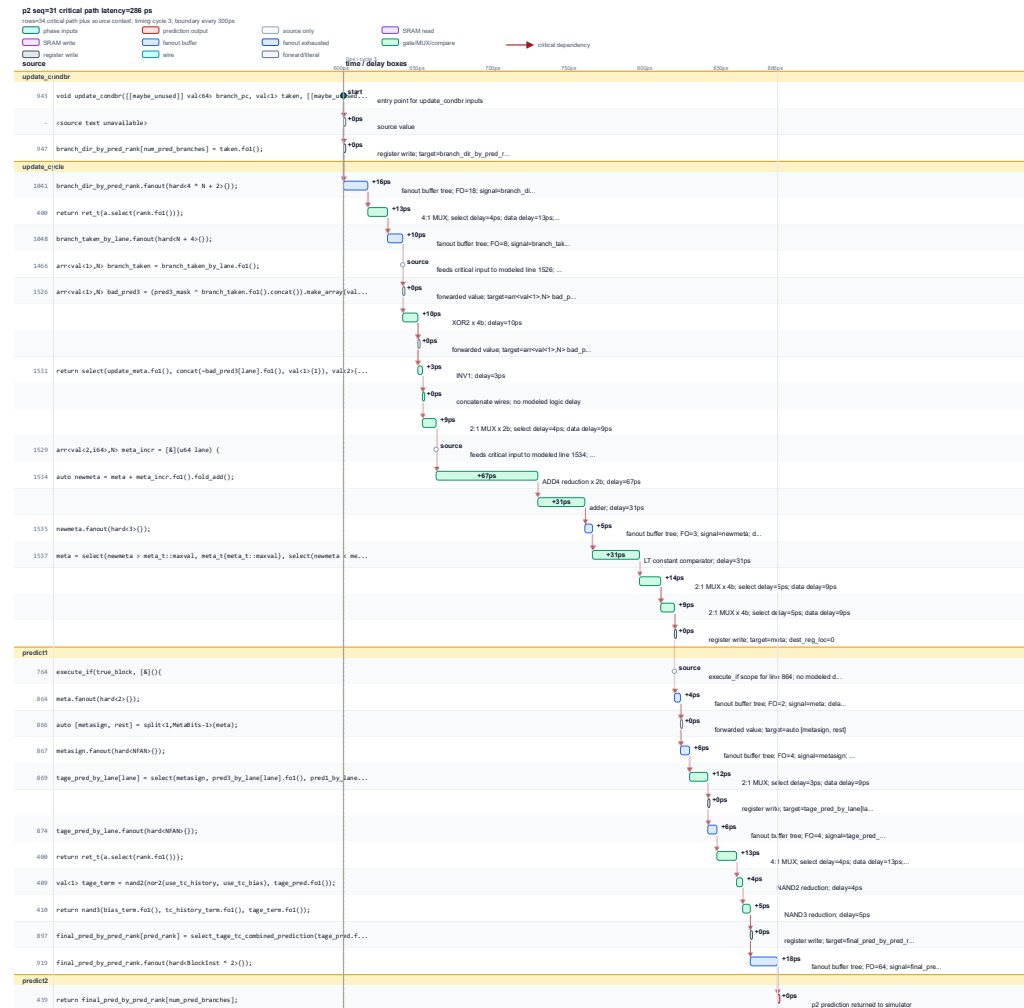
23/20



MORSL critical path

24/20

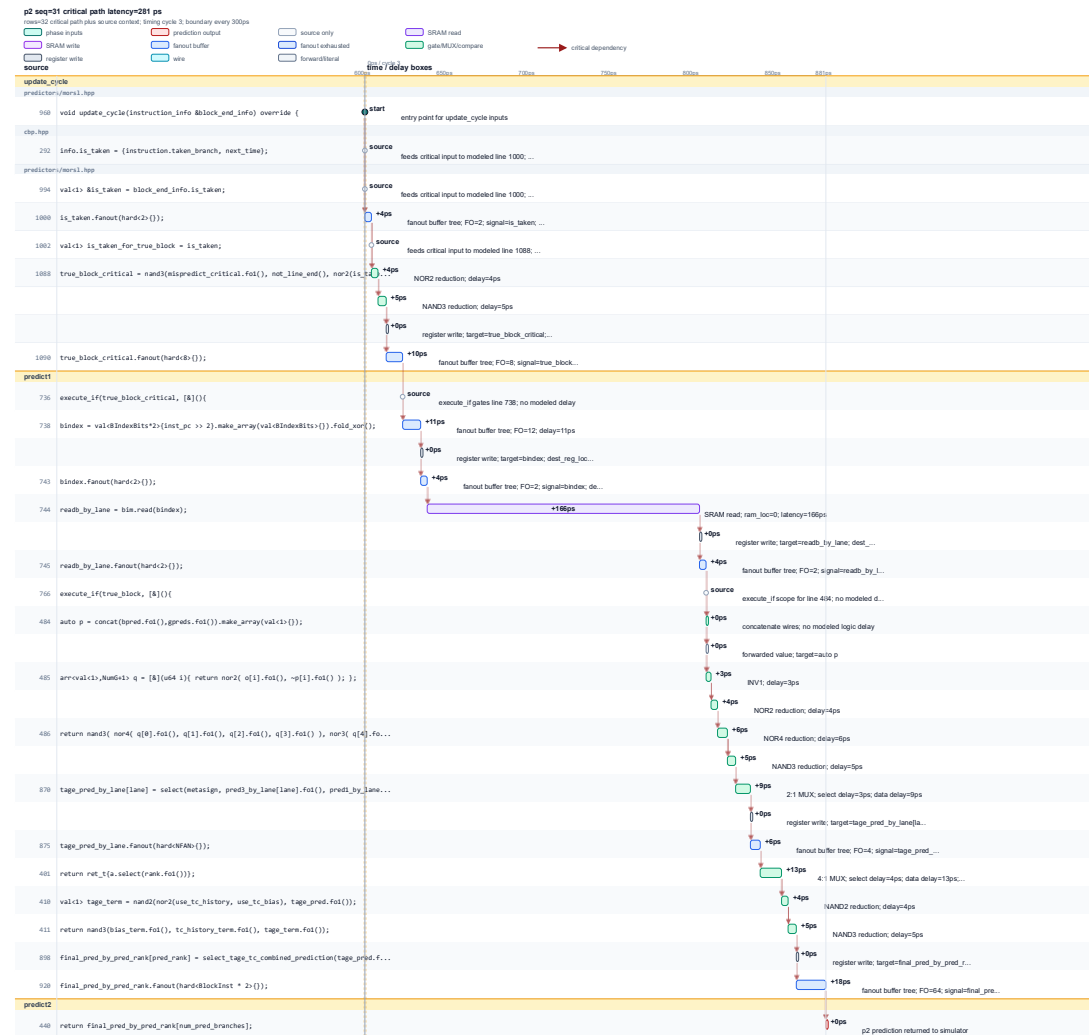
- 286ps
 - Updating meta predictor (can be pipelined)



MORSL critical path (2nd)

25/20

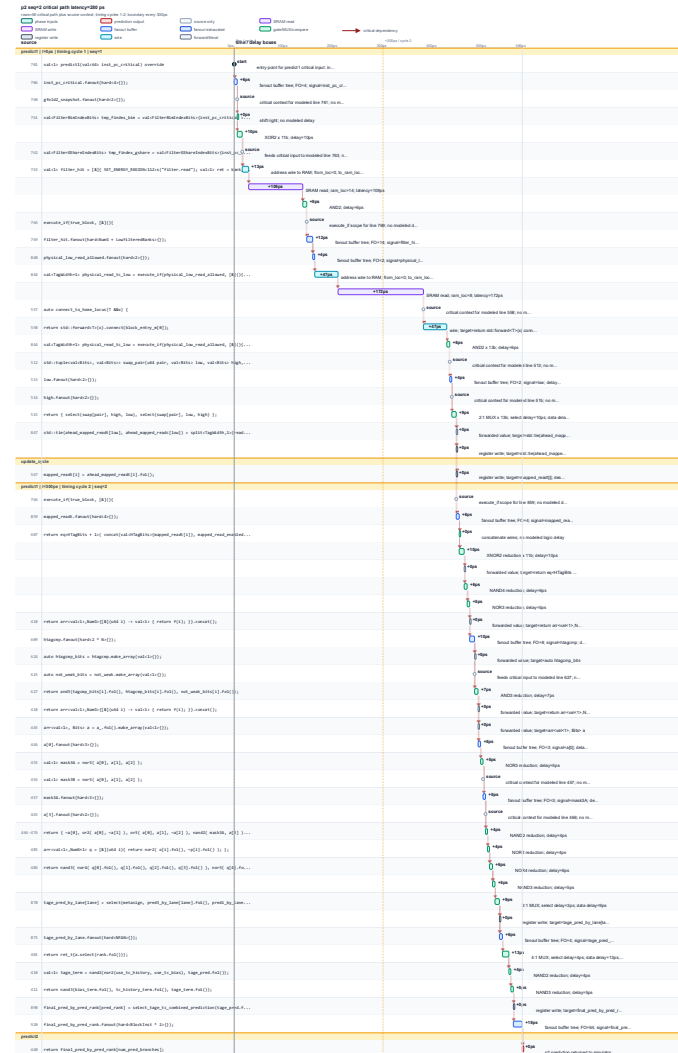
- 281ps
 - Bimodal



MORSL critical path (3rd)

26/20

- 580ps (2 pipeline stages)
 - Filter + Tagged tables



Future work: TC-history with sampler

27/20

- TC-history can handle multiple histories in one table!
 - It reduced MPKI by up to 0.02 without increasing main TC-history table entries.
 - We have not implemented in HARCOM.

